Zeppelin

FINISHED

```
%pyspark
from pandas import Series, DataFrame
import pandas as pd
import numpy as np

frame = DataFrame({'data1': np.random.randn(1000), 'data2': np.random.randn(1000)})
factor = pd.cut(frame.data1,4)
factor[:10]
```

```
0     (-0.0938, 1.475]
1    (-1.663, -0.0938]
2      (1.475, 3.0437]
3     (-0.0938, 1.475]
4     (-0.0938, 1.475]
5    (-1.663, -0.0938]
6     (-0.0938, 1.475]
7     (-3.238, -1.663]
8      (1.475, 3.0437]
9    (-1.663, -0.0938]
Name: data1, dtype: category
Categories (4, object): [(-3.238, -1.663] < (-1.663, -0.0938] < (-0.0938, 1.475] < (1.475,
3.0437]]
```

FINISHED

```
%pyspark
def get_stats(group):
  return {'min': group.min(), 'max': group.max(), 'count': group.count(), 'mean': group.me
grouped = frame.data2.groupby(factor)
grouped.apply(get_stats).unstack()
```

```
                   count       max      mean       min
data1
(-3.238, -1.663]    51.0  1.847021 -0.016305 -2.203226
(-1.663, -0.0938]  439.0  3.036204 -0.049544 -3.537838
(-0.0938, 1.475]   447.0  2.747356  0.055262 -3.404124
(1.475, 3.0437]     63.0  1.986052  0.005977 -3.107897
```

FINISHED

```
%pyspark
# compute quantile numbers
grouping = pd.qcut(frame.data1, 10, labels=False)
grouped = frame.data2.groupby(grouping)
grouped.apply(get_stats).unstack()
```

```
       count       max      mean       min
data1
0      100.0  2.372078 -0.064840 -3.537838
1      100.0  3.036204  0.060842 -2.300333
2      100.0  2.931313 -0.065915 -2.357223
3      100.0  2.939984 -0.091078 -2.821557
4      100.0  1.954144 -0.034637 -1.940848
5      100.0  2.181723 -0.076362 -2.512935
6      100.0  2.139447 -0.123829 -3.404124
7      100.0  2.366889  0.125820 -1.946764
8      100.0  2.747356  0.159315 -1.940647
9      100.0  2.095359  0.135655 -3.107897
```

FINISHED

```
%pyspark
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
s = Series(np.random.randn(6))
s[::2] = np.nan
s
```

```
0         NaN
1    1.114197
2         NaN
3   -0.402738
4         NaN
5    0.749553
dtype: float64
```

ERROR

```
%pyspark
s.fillna(s.mean())
```

FINISHED

```
%pyspark
states = ['Ohio', 'New York', 'Vermont', 'Florida', 'Oregon', 'Nevada', 'Califronia', 'Idal
group_key = ['East'] * 4 + ['West'] * 4
data = Series(np.random.randn(8), index=states)
data[['Vermont','Nevada','Idaho']] = np.nan
data
```

```
Ohio          0.742188
New York      0.416759
Vermont            NaN
Florida      -2.267450
Oregon       -0.935032
Nevada             NaN
Califronia    0.168355
Idaho              NaN
dtype: float64
```

FINISHED

```
%pyspark
data.groupby(group_key).mean()
```

```
East   -0.369501
West   -0.383338
dtype: float64
```

FINISHED

```
%pyspark
fill_mean = lambda g : g.fillna(g.mean())
data.groupby(group_key).apply(fill_mean)
```

```
Ohio          0.742188
New York      0.416759
Vermont      -0.369501
Florida      -2.267450
Oregon       -0.935032
Nevada       -0.383338
Califronia    0.168355
Idaho        -0.383338
dtype: float64
```

FINISHED

```
%pyspark
fill_values = {'East': 0.5, 'West': -1}
fill_func = lambda g: g.fillna(fill_values[g.name])
data.groupby(group_key).apply(fill_func)
```

```
Ohio          0.742188
New York      0.416759
Vermont       0.500000
Florida      -2.267450
Oregon       -0.935032
Nevada       -1.000000
Califronia    0.168355
Idaho        -1.000000
dtype: float64
```

FINISHED

```
%pyspark
```

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np

df = DataFrame({'category': ['a','a','a','a','b','b','b','b'], 'data': np.random.randn(8),
df
```

```
  category      data   weights
0        a  0.410116  0.378536
1        a -1.852803  0.336846
2        a  1.033562  0.800144
3        a  1.130160  0.615388
4        b  0.531768  0.985667
5        b  0.388601  0.749693
6        b -0.515092  0.900023
7        b -0.528385  0.986466
```

FINISHED

```
%pyspark
grouped = df.groupby('category')
get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
grouped.apply(get_wavg)
```

```
category
a    0.494445
b   -0.046758
dtype: float64
```

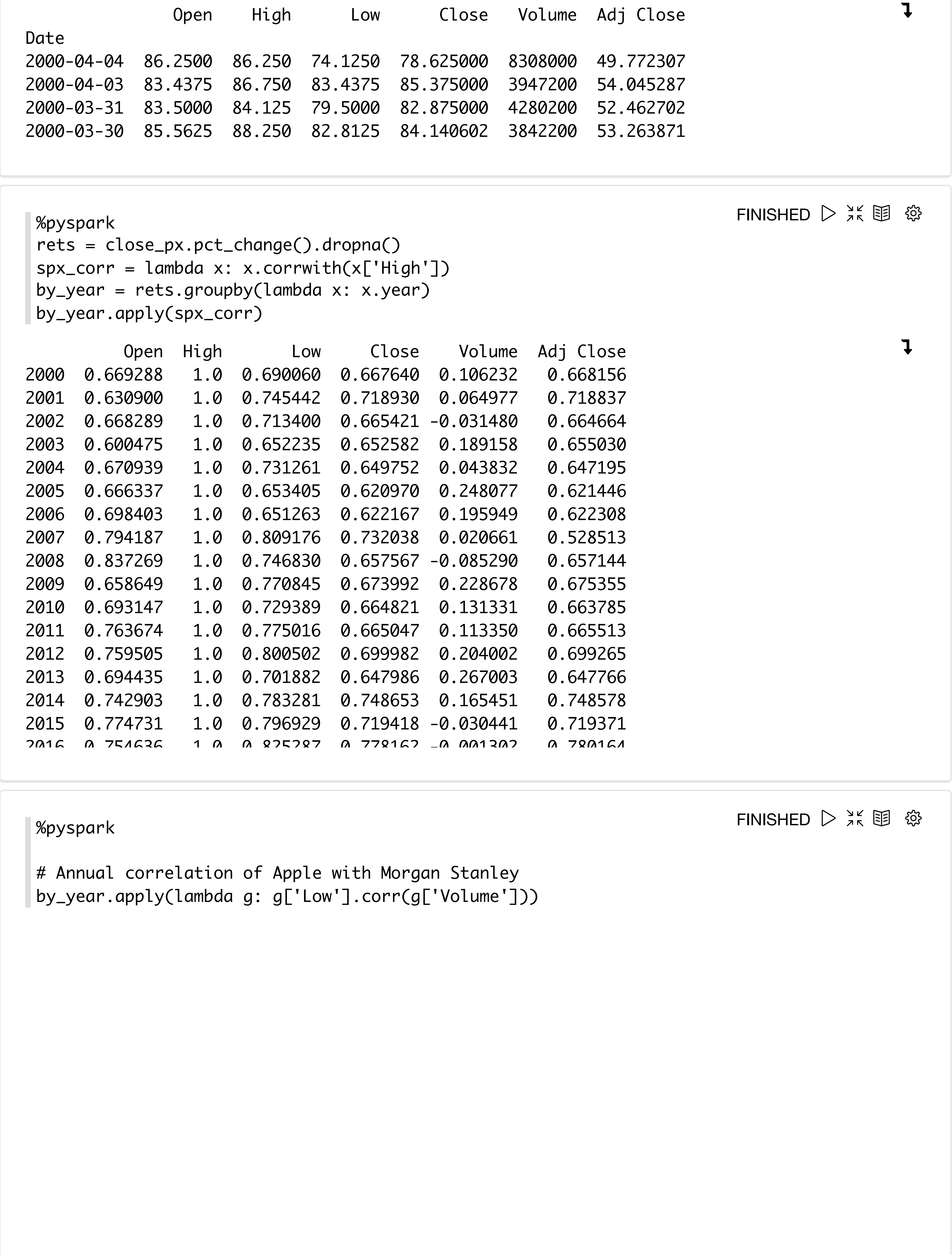
FINISHED

```
%pyspark
close_px = pd.read_csv('/Users/Rhon/Desktop/Capstone/MorganStanley.csv', parse_dates=True,
close_px.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4276 entries, 2017-03-29 to 2000-03-30
Data columns (total 6 columns):
Open         4276 non-null float64
High         4276 non-null float64
Low          4276 non-null float64
Close        4276 non-null float64
Volume       4276 non-null int64
Adj Close    4276 non-null float64
dtypes: float64(5), int64(1)
memory usage: 233.8 KB
```

FINISHED

```
%pyspark
close_px[-4:]
```



```
                Open    High      Low      Close   Volume  Adj Close
Date
2000-04-04  86.2500  86.250  74.1250  78.625000  8308000  49.772307
2000-04-03  83.4375  86.750  83.4375  85.375000  3947200  54.045287
2000-03-31  83.5000  84.125  79.5000  82.875000  4280200  52.462702
2000-03-30  85.5625  88.250  82.8125  84.140602  3842200  53.263871
```

```
%pyspark
rets = close_px.pct_change().dropna()
spx_corr = lambda x: x.corrwith(x['High'])
by_year = rets.groupby(lambda x: x.year)
by_year.apply(spx_corr)
```

```
          Open  High       Low     Close    Volume  Adj Close
2000  0.669288   1.0  0.690060  0.667640  0.106232   0.668156
2001  0.630900   1.0  0.745442  0.718930  0.064977   0.718837
2002  0.668289   1.0  0.713400  0.665421 -0.031480   0.664664
2003  0.600475   1.0  0.652235  0.652582  0.189158   0.655030
2004  0.670939   1.0  0.731261  0.649752  0.043832   0.647195
2005  0.666337   1.0  0.653405  0.620970  0.248077   0.621446
2006  0.698403   1.0  0.651263  0.622167  0.195949   0.622308
2007  0.794187   1.0  0.809176  0.732038  0.020661   0.528513
2008  0.837269   1.0  0.746830  0.657567 -0.085290   0.657144
2009  0.658649   1.0  0.770845  0.673992  0.228678   0.675355
2010  0.693147   1.0  0.729389  0.664821  0.131331   0.663785
2011  0.763674   1.0  0.775016  0.665047  0.113350   0.665513
2012  0.759505   1.0  0.800502  0.699982  0.204002   0.699265
2013  0.694435   1.0  0.701882  0.647986  0.267003   0.647766
2014  0.742903   1.0  0.783281  0.748653  0.165451   0.748578
2015  0.774731   1.0  0.796929  0.719418 -0.030441   0.719371
2016  0.754636   1.0  0.825287  0.778162 -0.001302   0.780164
```

```
%pyspark

# Annual correlation of Apple with Morgan Stanley
by_year.apply(lambda g: g['Low'].corr(g['Volume']))
```

```
2000   -0.263173
2001   -0.220835
2002   -0.262384
2003   -0.070999
2004   -0.309774
2005   -0.167750
2006   -0.228065
2007   -0.291658
2008   -0.322245
2009   -0.097808
2010   -0.209288
2011   -0.198386
2012   -0.069018
2013   -0.104618
2014   -0.163641
2015   -0.294724
2016   -0.215895
2017   -0.374866
```

FINISHED

```
%pyspark

# applying Ordinary Least Squares (OLS) regression on each chunk of data

import statsmodels.api as sm
def regression(data, yvar, xvars):
  Y = data[yvar]
  X = data[xvars]
  X['intercept'] = 1.
  result = sm.OLS(Y,X).fit()
  return result.params

by_year.apply(regression,'High',['Open'])
```

```
          Open  intercept
2000  0.562084   0.000418
2001  0.548357   0.000787
2002  0.556115   0.000468
2003  0.530821  -0.000462
2004  0.581484   0.000049
2005  0.574285  -0.000072
2006  0.564277  -0.000543
2007  0.701772   0.000531
2008  0.618472   0.001565
2009  0.569047  -0.000830
2010  0.547569   0.000237
2011  0.677649   0.000792
2012  0.615043  -0.000235
2013  0.639176  -0.000614
2014  0.645339  -0.000292
2015  0.649278   0.000250
2016  0.683428  -0.000341
```

FINISHED ▷

```
%pyspark
import timeit
start = timeit.timeit()
close_px[-4:]
end = timeit.timeit()
print(end - start)
```

```
0.00579380989075
```

FINISHED ▷

```
%pyspark
import timeit
start = timeit.timeit()

rets = close_px.pct_change().dropna()
spx_corr = lambda x: x.corrwith(x['High'])
by_year = rets.groupby(lambda x: x.year)
by_year.apply(spx_corr)

end = timeit.timeit()
print(end - start)
```

```
0.00370502471924
```

FINISHED ▷

```
%pyspark
import timeit
start = timeit.timeit()

by_year.apply(lambda g: g['Low'].corr(g['Volume']))

end = timeit.timeit()
print(end - start)
```

```
-0.00384306907654
```

FINISHED ▷

```
%pyspark
import timeit
start = timeit.timeit()

# applying Ordinary Least Squares (OLS) regression on each chunk of data

import statsmodels.api as sm
def regression(data, yvar, xvars):
  Y = data[yvar]
  X = data[xvars]
  X['intercept'] = 1.
  result = sm.OLS(Y,X).fit()
  return result.params

by_year.apply(regression,'High',['Open'])
```

```
end = timeit.timeit()
print(end - start)
```

```
-0.00264191627502
```

READY